Spring 6-12-2019

Fast Optimization Algorithm for Linear Regression Models with Sparsity-Inducing Penalties

Saket Pandit
saket.pandit@uconn.edu

Fast Algorithm for Linear Regression Models With Sparsity-Inducing Penalties

Saket Pandit

June 12, 2019

Abstract

This paper presents the FOS algorithm as first outlined by Lim and Lederer in 2016, and describes its intuition. FOS is an algorithm that efficiently traverses the L1 regularization path of the LASSO regression. This paper also presents a novel implementation of the FOS algorithm for the Group LASSO problem, and compares this algorithm against a state-of-the-art regression package. These algorithms both have potential for significant impact in biomedical researc regarding the analysis of gene expression data.

1 Introduction

This paper highlights the usage of an algorithm for reducing problems of high complexity into much simpler models that have increased interpretability. Two famous models used for reducing complexity are the Ridge and LASSO regressions. Both of these regressions are extensions of the Ordinary Least Squares regression that feature an additional penalty parameter. Each regression implements its penalty parameter in a unique way that is best utilized under particular circumstances.

1.1 Bias-Variance Tradeoff

Before one can understand what differentiates the Ridge and LASSO from the OLS regression, it is important to understand the Bias-Variance tradeoff. The Bias-Variance tradeoff is one that is prominent when one is deciding upon the best model to fit some data.

$$\mathsf{MSE}_{test} = \mathsf{E}(y_0 - \hat{f}(x_0))^2 = \mathsf{Var}(\hat{f}(x_0)) + \mathsf{B}(\hat{f}(x_0))^2 + \mathsf{Var}(\epsilon)$$

This is the formula for calculating test MSE, a value that is typically used to evaluate a given model, $\hat{f}(x_0)$ (James et al. 2017). The test MSE serves as a gauge for the efficacy of a model. It is calculated by adding the variance, the squared bias, and the natural variance of the model.

The variance of a model, $\mathsf{V}(\hat{f}(x_0))$ describes how much a model fluctuates when it is trained on different data. A high variance indicates that a model changes greatly when given new data, and a low variance indicates that a model does not fluctuate when given new data.

The other component of the test MSE is the squared Bias of a given model. The amount of bias that comes with a model illustrates how far this model is from the "true" relationship between variables. For example, a linear model used to describe a dataset with exponential growth would incur a high bias.

Generally, as the bias decreases across models, the variance increases. An OLS regression is on one end of the spectrum of models, in that it is a model with generally low variability and high bias. However, when dealing with large datasets, the OLS regression typically generates a higher variance as the number of predictors increases. This is where the Ridge and LASSO regressions step in.

1.2 Ridge Regression

Both the Ridge regression and the LASSO regression are a part of a family of regressions that make use of a penalty parameter to influence the way that the data is fit. This penalty parameter is put alongside the Ordinary Least Squares (OLS) regression, and both of these components work in tandem to produce a model.

OLS Regression

$$L^{OLS}(\beta_1, \ldots, \beta_p) = \sum_{i=1}^{n} (y_i - \sum_{j=1}^{p} (x_{ij}\beta_j))^2$$

This is the loss function for the OLS regression.

Ridge Regression

$$L^{Ridge}(\beta_1, \ldots, \beta_p) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{p} (x_{ij}\beta_j))^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

This is the loss function for the Ridge regression. Juxtaposing the Ridge regression with the OLS regression makes the similarities between the two very clear, as the Ridge regression is essentially the OLS regression with a penalty parameter added on (the $\frac{1}{2n}$ is negligible, since this is a constant). The penalty parameter allows the Ridge regression to shrink all the variables used in a model towards 0, which is why the Ridge regression is part of the shrinkage regressions family. Shrinkage is especially useful when dealing with large datasets with a high number of predictors - also known as *high-dimensional datasets* - as it helps to decrease the variance of the OLS regression. Specifically, the Ridge regression helps to reduce the variance by decreasing the influence of certain variables on predicting the response. In models such as the Ridge or the LASSO, the regression implements penalties on variables of lower influence.

In the Ridge regression, the way that this penalty is implemented results in a reduction of the magnitude of the coefficients. This leads to a better explanation of the data than what one would obtain from an OLS regression run on the same dataset. This is because in an OLS regression, the same weight is placed on all variables, regardless of their influence. In the Ridge regression, higher weights are placed on variables that can better explain the data and lower weights are placed on variables that cannot. This yields a more accurate model than that from the OLS regression.

1.3 LASSO Regression

$$L^{LASSO}(\beta_1, \ldots, \beta_p) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \sum (x_{ij}\beta_j))^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

This is the loss function for the LASSO regression (R. Tibshirani 1996). The LASSO regression also makes use of a penalty parameter. The key difference between the penalty of the LASSO regression and that of the Ridge regression is that the LASSO penalty is able to set the non-influential variables equal to 0 instead of only shrinking them. Setting the coefficient of a variable to 0 discards the variable from the final model. This is why the LASSO regression is considered a variable selection method, because the final model selects the variables that are the most influential.

While the LASSO regression is also useful for high-dimensional datasets, it can be used for a specific subset of high-dimensional datasets: sparse data. Sparse datasets are high-dimensional datasets in

which the vast majority of the predictors are non-influential. Thus, LASSO will directly tell the user which variables are of the highest influence.

1.4 Synthetic Datasets

Synthetic datasets, also known as "toy data" are manufactured datasets that are used to assess the efficacy of machine learning algorithms. Toy data are more useful than "real" datasets because they are generated with a set of known parameters. Thus, since we know what the true values of β are, we can easily know how closely our model comes to estimating these coefficients.

This paper will make use of toy data to demonstrate the efficacy of the machine learning algorithms presented. For the LASSO regression, the toy data include a 300×100 matrix of observations, and a 100-element vector of responses. The responses are calculated beforehand using a combination of β, the vector containing the true coefficients that will be estimated, and random noise generated from a Gaussian distribution.

Additionally, a grid of λ tuning parameters are required in the LASSO regression. For the toy data, these tuning parameters start from the maximum λ value and go down to a fraction of this value. The maximum λ value is calculated through the following equation: $\lambda_{max} = \frac{\|X^T y\|_\infty}{n}$

2 Methods

2.1 LASSO

2.1.1 The Loss Function

$$\frac{1}{2n}\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p}(x_{ij}\beta_j))^2 + \lambda\sum_{j=1}^{p}|\beta_j|$$

The loss function, as was described in an earlier section, is a tool used in machine learning. The loss function is a function of β that describes the error associated with β, for a given X (explanatory variable) and y (response variable). The loss function is what is used to fit a model to the data. After evaluating the loss function, we can arrive at a model that uses the coefficients that most accurately predict the response. One way that this is done is through optimization of the loss function.

2.1.1.1 LASSO Penalty

Before we can optimize the loss function, it is important to understand the significance of the penalty parameter utilized in the LASSO regression. The penalty parameter can be broken down into two components: the lambda parameter and the L1 regularization of the coefficients.

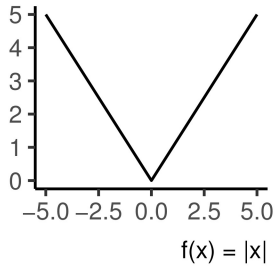
The lambda parameter is essentially a tuning parameter for the amount of penalty induced by the model; too high a value sets all the coefficients equal to 0, and too low a value leaves too many uninfluential variables in the final model. Therefore, usually cross-validation is required to determine the best lambda value for a given regression.

The L1 regularization of the coefficients is what actually sets the penalty on the coefficients. This penalty placed on the coefficients becomes apparent in the gradient of the loss function.

2.2 Optimization Strategies

2.2.1 Optimizing for LASSO: The Soft Threshold

In order to find the coefficients that best explain the data, we must optimize the LASSO loss function with respect to β in order to find the optimal $\tilde{\beta}$ that fits the data. The optimization of the LASSO regression involves calculation of the gradient of the function. However, there is one caveat in taking the gradient of the LASSO loss function: the L1 norm. The L1 norm involves the use of the absolute value, a non-differentiable function. There are mathematical shortcuts that can be used to go around this issue.



One such shortcut can be best explained through an analysis of a function that makes use of the absolute value, such as $f(x) = |x|$. One can see that before $x = 0$ the slope of the line is -1, and after $x = 0$ the slope of the line is 1. However, when $x = 0$ the derivative does not exist, which makes $x = 0$ a non-differentiable point in this function. This makes the gradient of the LASSO function difficult to arrive at. Therefore, we must use the sub-gradient in order to optimize the LASSO function. Generally, the sub-gradient is used to generalize a gradient at a non-differentiable point. In this case, we make use of the soft-thresholding operator for the sub-gradient of the L1 norm. The soft-thresholding operator assumes that the value of the gradient when $x = 0$ as in the above function must be in $[-1, 1]$.

When the soft-thresholding operator is applied to a function, it typically assumes the following form (T. Hastie, Tibshirani, and Wainwright 2015):

$$\hat{\theta} = \arg\min_{\theta \in \mathbb{R}} \left\{ \frac{1}{2}(\theta - z)^2 + \lambda|\theta| \right\} = \begin{cases} z - \lambda, & \text{if } z > \lambda \\ 0, & \text{if } -\lambda \leqslant z \leqslant \lambda \\ z + \lambda, & \text{if } z < -\lambda \end{cases} = S(z; \lambda)$$

Therefore, applying the soft-thresholding operator to the LASSO problem yields:

$$\tilde{\beta}_j = \begin{cases} \tilde{z}_j - \frac{n\lambda}{\sum_{i=1}^n x_{ij}^2}, & \text{if } \tilde{z}_j > \frac{n\lambda}{\sum_{i=1}^n x_{ij}^2} \\ 0, & \text{if } \frac{-n\lambda}{\sum_{i=1}^n x_{ij}^2} \leqslant \tilde{z}_j \leqslant \frac{n\lambda}{\sum_{i=1}^n x_{ij}^2} \\ \tilde{z}_j + \frac{n\lambda}{\sum_{i=1}^n x_{ij}^2}, & \text{if } \tilde{z}_j < \frac{n\lambda}{\sum_{i=1}^n x_{ij}^2} \end{cases} = \mathcal{S}\left(\tilde{z}_j; \frac{n\lambda}{\sum_{i=1}^n x_{ij}^2}\right)$$

where $\tilde{z}_j = \frac{1}{\sum_{i=1}^n x_{ij}^2} \sum_{i=1}^n x_{ij} r_i + \beta_j$

The soft-thresholding operator sets bounds for z_j that are a function of values generated from the dataset and λ. Notice that z_j is a function of a β_j value. If z_j falls wtihin these bounds, then the β_j is then set to 0. β_j is calculated through an optimization algorithm applied to the LASSO problem. Three of these algorithms are described in the following sections.

It is important to note that while the LASSO regression is convex, it is not *strongly* convex. One of the features of a convex problem is that the local minimum is also the global minimum. However, because the LASSO is not strongly convex, it is possible to have multiple global minimum. Therefore, it is possible to have multiple unique solutions to any given LASSO problem.

2.2.2 Gradient Descent

Gradient descent works through the calculation of the gradient of the LASSO loss function. Because the optimization of the function is with regards to a $\mathbf{j}$-length vector of β, this problem is one with $\mathbf{j}$ dimensions. Once this gradient is calculated, the algorithm proceeds in the direction opposite the gradient. The magnitude at which gradient descent proceeds is determined by the α value. This value is also of vital importance, because choosing the right α value can generate a solution. The α value determines how big of a "step" the β values must take in order to descend down the loss function. If too large a step is taken, the optimal solution might be skipped over, leading to non-convergence. If too small a step is taken, then while the algorithm may reach the optimal solution it will be computationally taxing. Therefore, some trial-and-error is necessary before landing on an ideal α value. Gradient descent ends when a local optimum is found.

Gradient descent can be implemented as in **Algorithm 1**.

Algorithm 1: Gradient Descent Algorithm for LASSO

Inputs : data (Y, X), cross-validated λ_M
Output: estimate regression vector $\hat{\beta}(\lambda_M)$

randomize betaOld $\sim N(0, 1)$;
$\alpha = 0.05$;
cont = TRUE;
$\epsilon = 2 \times 10^{-6}$;
for $s = 0, 1, \ldots$ **do**
  while *cont* = *TRUE* **do**
    Calculate OLS $= -\frac{1}{n} X^T(y - X\text{betaOld})$;
    for $j = 1, \ldots, p$ **do**
      Calculate penalty $= \mathcal{S}(\text{betaOld}_j; \lambda_m)$;
    end
    Calculate ∇ = OLS + penalty;
    Update betaNew = betaOld $- \alpha\nabla$;
    Update cont $> \|\text{betaNew} - \text{betaOld}\|_2$;
    betaNew := betaOld;
  end
end
if *cont* = *FALSE* **then**
  break
end

2.2.3 Coordinate Descent

$$\tilde{\beta}_j = \arg\min_{\beta_j \in R} \left\{ L(\beta) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \sum_{k \neq j} x_{ij} \beta j)^2 + \lambda |\beta| \right\}$$

This is the problem that Coordinate descent looks to solve. Coordinate descent works in a very similar fashion as gradient descent, except for two key differences. The first difference is that instead of taking the whole gradient of the loss function, coordinate descent uses the partial gradient for one value of j, the number of parameters, at a time. This is useful especially in problems with a high number of predictors because of its computational efficiency. It is much more difficult to calculate the gradient of a 100-dimensional problem than it is to calculate a 1-dimensional partial gradient. The partial gradients are calculated in an iterative process, and the β values are continuously updated. The descent proceeds in a direction determined by the negative partial gradient, which is similar to gradient descent.

However, when coordinate descent stops is different from gradient descent. Coordinate descent stops when the change made to the coefficients, the vector β, goes below a preset threshold. This is opposed to gradient descent, which guarantees convergence at a local optimum. The stopping criteria for coordinate descent, on the other hand, provides no such guarantee. However, in practice coordinate descent and gradient descent often yield the same or similar solutions. For dealing with our toy data, coordinate descent is the more effective algorithm, so that is what is used for the rest of this paper.

Coordinate descent can be implemented as in **Algorithm 2**.

Algorithm 2: Coordinate Descent Algorithm for LASSO

Inputs : data (Y, X), sequence of M tuning parameters $\lambda_1 = \lambda_{max} > \ldots > \lambda_M > 0$
Output: estimate regression vectors $\hat{\beta}(\lambda_1), \ldots, \hat{\beta}(\lambda_M)$

$\hat{\beta}(\lambda_1) = 0$;
$r_i = y_i$ for all $i = 1, \ldots, n$;
for $m = 2, \ldots, M$ **do**
  $\tilde{\beta}^{(0)}(\lambda_m) = \hat{\beta}(\lambda_{m-1})$;
  for $s = 0, 1, \ldots$ **do**
    for $j = 1, \ldots, p$ **do**
      Calculate $\tilde{z}_j = \sum_{i=1}^{n} x_{ij} r_i / \sum_{i=1}^{n} x_{ij}^2 + \tilde{\beta}_j^{(s)}(\lambda_m)$;
      Update $\tilde{\beta}_j^{(s+1)} = \mathcal{S}\big(\tilde{z}_j; n\lambda_m / \sum_{i=1}^{n} x_{ij}^2\big)$;
      Update $r_i := r_i - (\tilde{\beta}_j^{(s+1)}(\lambda_m) - \tilde{\beta}_j^{(s)}(\lambda_m)) x_{ij}$ for all $i = 1, \ldots, n$;
    end
    if *stopping criterion met* **then**
      $\hat{\beta}(\lambda_m) = \tilde{\beta}^{(s)}(\lambda_m)$;
      break;
    end
  end
end

2.2.4 Dual Formulation of LASSO

Another optimization strategy is the duality optimization. The duality optimization is concerned with two different-yet-related functions: the primal problem and the dual problem. The primal problem is the original optimization problem, in this case the LASSO problem:

$$\arg\min_{\beta_j \in R} \{L(\beta, \lambda) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \sum_{k \neq j} x_{ij} \beta j)^2 + \lambda |\beta|\}$$

The dual problem is created from the primal problem using the gradient of the LaGrangian transformation of the primal problem. One of the special properties of the dual problem is that the maximum value of the dual problem is the minimum value of the primal problem. In the case of the LASSO, the dual problem is as follows:

$$\arg\max_{\nu \in \mathbb{R}^n} \{D(\nu, \lambda) = -\lambda^2 \|\nu + 2Y/\lambda\|_2^2/4 + \|Y\|_2^2\}$$

Duality optimization requires the calculation of a duality gap, which is the difference between the primal and dual formulations. In calculating the duality gap, we are showing how far apart the dual problem and the primal problem are from one another. If the maximum of the dual problem and the minimum of the primal problem are equivalent, then at the optimal solution of the LASSO problem the duality gap would equal 0. Therefore, if we set a threshold value for the duality gap very close to zero, we will know that once the duality gap passes this threshold that we are very close to the optimal solution.

2.3 FOS

2.3.1 General Intuition

The FOS strategy was developed by Néhémy Lim and Johannes Lederer in "Efficient Feature Selection With Large and High-dimensional Data." (Lim and Lederer 2016). FOS descends the LASSO loss function using a combination of coordinate descent and the dual formulation of the LASSO problem. FOS also implements a unique threshold value that is distinct from the one used in coordinate descent. In coordinate descent, a threshold is set, and the algorithm stops once the difference between the predicted β_{j-1} and β_j dips below this threshold. As was mentioned earlier, this algorithm provides no guarantees that it will reach a local optimum but only that it will reach an area where the gradient is not steep.

On the other hand, the FOS strategy abandons this threshold in favor of one constructed in tandem with the λ value. By using a combination of the two, FOS creates a dynamic stopping threshold. For higher values of λ, the threshold is greater in magnitude and thus is less restrictive. As the λ values decrease, the threshold becomes increasingly restrictive and more steps of coordinate descent are required for the duality gap to dip below this threshold.

Through the use of a dynamic stopping threshold, FOS traverses the regularization path in a very efficient manner. It is already known that higher values of lambda will yield no non-zero coefficients, so it is not necessary to expend computing power to calculate these coefficients. Lower values of lambda will yield many non-zero coefficients, so this part of the regularization path is also not of interest to the user. The area between these two is what is of interest to a user, and FOS is able to wade through these intermediary values of λ in a precise manner. FOS then produces a single $\hat{\beta}$, and its associated λ, without the need for cross-validation, a time-consuming and compute-intensive process.

One can implement FOS as is outlined by **Algorithm 3**.

2.4 Group LASSO

2.4.1 General Intuition

If it is known a priori that the variables of a dataset can be separated into disjoint groups, and that only few of these groups are influence the response, then the Group LASSO becomes a powerful tool

Algorithm 3: FOS Algorithm for LASSO

Inputs : $Y \in \mathbb{R}^n; X \in \mathbb{R}^{n\times p}; {}_{-1} = {}_{\max} > \ldots > {}_{M} > 0; , , c_{stat}, c_{comp} > 0$
Output: $\tilde{\beta}^{\tilde{\lambda}} \in \mathbb{R}^p; \tilde{S} \subset \{1, \ldots, p\}$

Initialization: statsCont:=TRUE; statsIt:=1; $\tilde{\beta}^{\lambda_1}$:=0; $\tilde{\lambda} := \lambda_M$;
while *statsCont==TRUE and statsIt<M* **do**
  statsIt := statsIt + 1;
  stopCrit := FALSE;
  betaOld := $\tilde{\beta}^{\lambda_{\text{statsIt}-1}}$;
  while *stopCrit == FALSE* **do**
    Compute a dual feasible point $\tilde{\nu}^{\lambda_{\text{statsIt}}}$;
    Compute the duality gap $\Delta(\tilde{\beta}^{\lambda_{\text{statsIt}}}, \tilde{\nu}^{\lambda_{\text{statsIt}}})$;
    if $\Delta(\tilde{\beta}^{\lambda_{statsIt}}, \tilde{\nu}^{\lambda_{statsIt}}) \leqslant c^2_{comp}\lambda^2_{statsIt}/n$ **then**
      $\tilde{\beta}^{\lambda_{\text{statsIt}}}$:= betaOld stopCrit := TRUE
    else
      for $j = 1, \ldots, p$ **do**
        Calculate $\tilde{z}_j = \sum_{i=1}^n x_{ij} r_i / \sum_{i=1}^n x_{ij}^2 + \text{betaOld})$;
        Update $\tilde{\beta}^{\lambda_{\text{statsIt}}} = S\big(\tilde{z}_j; n\lambda_m / \sum_{i=1}^n x_{ij}^2\big)$;
        Update $r_i := r_i - (\tilde{\beta}^{\lambda_{\text{statsIt}}} - \text{betaOld})x_{ij}$ for all $i = 1, \ldots, n$;
      end
    end
  end
  statsCont := $\prod_{k=1}^{\text{statsIt}} \mathbb{1}\{\|\tilde{\beta}^{\lambda_{\text{statsIt}}} - \tilde{\beta}^{\lambda_k}\|_\infty / (\lambda_{\text{statsIt}} + \lambda_k) - (c_{\text{stat}} + c_{\text{comp}}/\sqrt{\gamma})/n\}$
end
if *statsCont==FALSE* **then**
  $\tilde{\lambda} := \lambda_{\text{statsIt}-1}$
end
$\tilde{S} := \{j \in \{1, \ldots, p\} : |\tilde{\beta}^{\tilde{\lambda}}_j| \geqslant 3(c_{\text{stat}} + c_{\text{comp}}/\sqrt{\gamma})\tilde{\lambda}/n\}$

for the selection of these influential groups. Group LASSO is a special implementation of the LASSO regression that acts on groups of variables instead of on individual variables.

Take, for example, a dataset containing the expression levels of 1000 genes as they relate to some disease. If it can be shown through a review of literature that these 1000 genes can be partitioned into 100 groups of genes, group LASSO can show the user which of these groups of genes are most influential in predicting the presence of the disease. Computationally, this method is more efficient in datasets that can be partitioned into groups. Instead of calculating a β vector of size 1000, we only have to compute a vector of size 100 (one coefficient for each group rather than one coefficient for each variable).

2.4.2 Loss function

$$L^{\text{GroupLASSO}}(\beta_1, \ldots, \beta_p) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda\sum_{j=1}^{k}\sqrt{p_j}\|\beta_{G_j}\|_2$$

Where p_j is the number of predictors in group G_j (Yuan and Lin 2006)

This is the loss function for the group LASSO. It is only slightly modified from that of the LASSO. While the LASSO features the L1 regularization of the calculated coefficients, the group LASSO utilizes

the L1/L2 regularization of the parameters. The L1/L2 regularization is necessary because of the partitioning element present in these kinds of problems. L1/L2 regularization takes the maximum value out of all the L2 norms of each group of coefficients. The L1/L2 regularization is what allows the group LASSO to assign weights to each group of coefficients, and then to set these weights to 0 if the group is not sufficiently influential.

2.4.3 Block coordinate descent

Block coordinate descent is very similar to coordinate descent. While coordinate descent cycles through the individual β_j's in a problem, block coordinate descent cycles through the groups. Block coordinate descent recognizes each of the groups of variables as separate blocks, and then assigns a weight β_w to each of these blocks. If the block is non-influential, then this β_w is set to 0, and therefore all the $\mathtt{beta}_j$'s within that group are also set to 0. Block coordinate descent utilizes the gradient of the group LASSO function which, like the LASSO, also requires a soft-thresholding operator as a sub-gradient. For block coordinate descent, we use the block soft-thresholding operator:

$$\hat{\theta} = \arg\min_{\theta \in R^v} \{\frac{1}{2}\|\theta - z\|_2^2 + \lambda\|\theta\|_2\} = \mathcal{BST}(z;\lambda)\}$$

2.4.4 Toy data

Due to the added partitioning element present in problems involving the Group LASSO, adjustments need to be made to the toy data. For evaluating the performance of the group LASSO, the previous data involving the observations and response values will be kept intact. Along with this, group LASSO requires a list of partitions for the coefficients. These partitions must be non-overlapping, meaning any given coefficient cannot be in more than 1 group.

Furthermore, the formula for calculating the maximum λ value has to be adjusted to the following: $\lambda_{max} = \frac{\max_{j=1,\ldots,k} \|X^T_{:,G_j} y\|_2/\sqrt{p_j}}{n}$

2.4.5 Group LASSO and FOS

This paper presents a unique implementation of group LASSO and FOS. In the 2016 paper from Lim (Lim and Lederer 2016) that introduces the FOS algorithm, the framework is used to solve the LASSO problem. **Algorithm 4** illustrates how one can use the FOS algorithm to solve the group LASSO problem.

3 Results

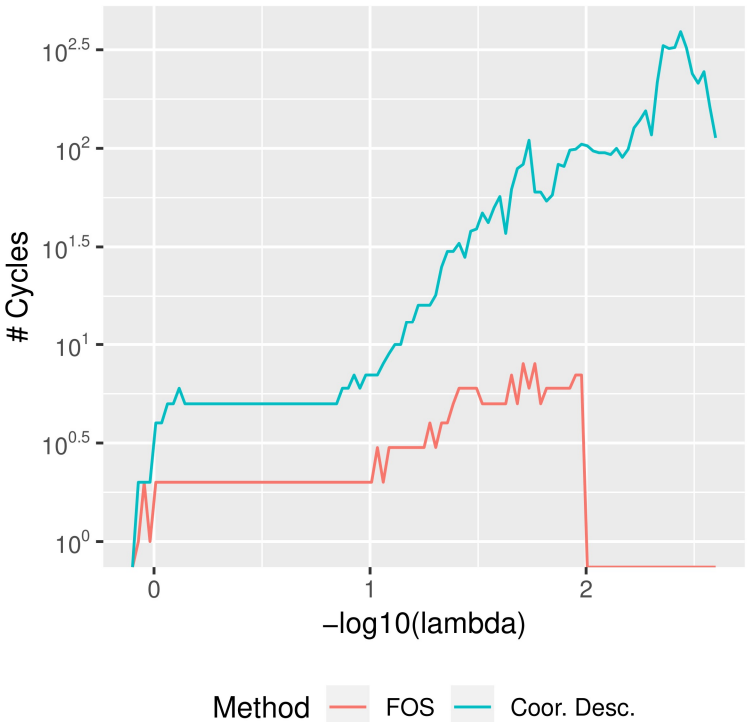
3.1 FOS and Coordinate Descent

As was mentioned in an earlier section, one of the advantages of using a dynamic threshold, like in FOS, over the use of a static treshold, like in coordinate descent, is that this is computationally more efficient. This computational efficiency is evident in the following figure, depicting the nunmber of cycles used in both algorithms:

Algorithm 4: FOS Algorithm for Group LASSO

Inputs : $Y \in \mathbb{R}^n; X \in \mathbb{R}^{n\times p}; {}^{-}_{1} = {}^{-}_{\max} > \ldots > {}^{-}_{M} > 0; , , c_{stat}, c_{comp} > 0$
Output: $\tilde{\beta}^{\tilde{\lambda}} \in \mathbb{R}^p; \tilde{S} \subset \{1, \ldots, p\}$

Initialization: statsCont:=TRUE; statsIt:=1; $\tilde{\beta}^{\lambda_1}$:=0; $\tilde{\lambda} := \lambda_M$;
while *statsCont==TRUE and statsIt<M* **do**
  statsIt := statsIt + 1;
  stopCrit := FALSE;
  betaOld := $\tilde{\beta}^{\lambda_{\text{statsIt}-1}}$;
  while *stopCrit == FALSE* **do**
    Compute a dual feasible point $\tilde{\nu}^{\lambda_{\text{statsIt}}}$;
    Compute the duality gap $\Delta(\tilde{\beta}^{\lambda_{\text{statsIt}}}, \tilde{\nu}^{\lambda_{\text{statsIt}}})$;
    if $\Delta(\tilde{\beta}^{\lambda_{statsIt}}, \tilde{\nu}^{\lambda_{statsIt}}) \leqslant c^2_{comp}\lambda^2_{statsIt}/n$ **then**
      $\tilde{\beta}^{\lambda_{\text{statsIt}}}$:= betaOld stopCrit := TRUE
    else
      for $j = 1, \ldots, p$ **do**
        Calculate $\tilde{z}_{G_j} = X^T_{:,G_j}\lambda_m/\|X_{:,G_j}\|^2_2 + \text{betaOld}_{G_j}$;
        Update $\tilde{\beta}^{\lambda_{\text{statsIt}}}_{G_j} = \mathcal{BST}\big(\tilde{z}_{G_j}; n\lambda_m\sqrt{p_j}/\|X_{:,G_j}\|^2_2\big)$;
        Update $r := r - X_{:,G_j}(\tilde{\beta}^{\lambda_{\text{statsIt}}}_{G_j} - \text{betaOld}_{G_j})$;
      end
    end
  end
  statsCont := $\prod_{k=1}^{\text{statsIt}} \mathbb{1}\{\|\tilde{\beta}^{\lambda_{\text{statsIt}}} - \tilde{\beta}^{\lambda_k}\|_\infty/(\lambda_{\text{statsIt}} + \lambda_k) - (c_{\text{stat}} + c_{\text{comp}}/\sqrt{\gamma})/n\}$
end
if *statsCont==FALSE* **then**
  $\tilde{\lambda} := \lambda_{\text{statsIt}-1}$
end
$\tilde{S} := \{j \in \{1, \ldots, p\} : |\tilde{\beta}^{\tilde{\lambda}}_j| \geqslant 3(c_{\text{stat}} + c_{\text{comp}}/\sqrt{\gamma})\tilde{\lambda}/n\}$



This graph shows the number of cycles required by a traditional coordinate descent algorithm and that required by FOS. As the value of λ decreases, the cycles for both coordinate descent and FOS increase. Note that even at the higher values of λ, FOS does not require as many cycles to proceed as coordinate descent. The most important part of this graph is at the end. The number of cycles that FOS requires for the lower values of λ drops to 0, while the coordinate descent algorithm continues to calculate coefficients for these values of λ. This is significant because it is known that these lower values of λ will yield many non-zero coefficients, so it is not of interest to a user who is searching for the most influential coefficients. Traditional coordinate descent requires a user to run a cross-validation procedure in order to arrive at an ideal value of λ, whereas FOS does not require the user to do so.

3.2 FOS empirical performance

Method	User time	System time	Elapsed time	Hamming distance	Test MSE
FOS	1.10	0.04	1.14	7	0.9698
glmnet	0.25	0.02	0.26	11	0.7948

This table compares the performance of FOS as compared to a state-of-the-art algorithm, **glmnet**, an algorithm that uses coordinate descent (Friedman, Hastie, and Tibshirani 2010). In this table, the performance is evaluated through several measures. The first three measures are timings: User, System, and Elapsed times. The User and System times are concerned with the amount of time taken by the CPU to call and execute the function. The Elapsed time is the total amount of time that the function takes. The next two measures focus on the accuracy of the regression. The Hamming distance is essentially the total number of false positives and false negatives produced by the regression. In this case, the Hamming distance is based on how close each regression is to predicting the true coefficients, β, with a lower Hamming distance meaning that the regression is closer to the truth. This is easy to calculate in this setting due to the use of toy data. The last measure is the Test Mean Squared Error, or MSE_{test}. This value is calculated through a cross-validation procedure. This means that the original data is split into a training set and a test set. The training set is used to build the model, while the test set is used to evaluate how well the model fits new values that it has not yet encountered. A lower MSE_{test} is indicative of a model with a better fit.

Looking at this table, it is clear that **glmnet** outperforms FOS in the timing measures and it has a lower MSE_{test} value. Even though **glmnet** has faster timings than FOS, it is important to remember that this package is written in Fortran, while FOS is written in R. Fortran is a low-level language which speaks almost directly to the computers, while R is a relatively high-level language that is more user-friendly. Because of this, there is some additional compute time that is spent interpreting the code written in R that is not present in the **glmnet** implementation of coordinate descent. Therefore, it is unclear how much **glmnet** gains from being written in a lower-level language as far as timing is concerned. Furthermore, regarding the MSE_{test} value for **glmnet**, the way that the coefficients are produced requires **glmnet** to run a cross-validation procedure that selects for the λ parameter that produces the lowest MSE value. Therefore, it makes sense that the coefficients produced by this λ value would yield a low MSE_{test} value.

In spite of these performance measures, FOS outperforms **glmnet** in the Hamming distance measure. This shows the merit of the FOS algorithm in that it is highly accurate in discerning which variables are the most influential in predicting a response. This has implications for future implementations of the algorithm.

3.3 Group LASSO FOS empirical performance

Method	User time	System time	Elapsed time	Hamming distance	Test MSE
Group FOS	1.03	0	1.04	0	0.9589
grpreg	0.44	0	0.44	1	0.5275

This table compares the performance the group LASSO implementation of FOS with **grpreg**, an algorithm that uses group descent (Breheny and Huang 2013) and is popular in machine learning. Like the previous table, this compares the performance of group FOS and **grpreg** as measured by User, System, and Elapsed times, as well as Hamming distance and MSE_{test}. One important note is that due to the nature of these regressions, the Hamming distance is calculated differently. Here, instead of showing how far each regression is in predicting the truly influential coefficients, the Hamming distance instead measures how far each regression is in predicting the truly influential groups.

Looking at this table, it is clear that **grpreg** outperforms FOS in the timing measures and it has a lower MSE_{test} value. As was the case in the previous comparison, the group LASSO FOS is slower than **grpreg**. However, while the group LASSO FOS was written in R, **grpreg** was written in C, another low-level language. It is unclear how much **grpreg** gains in computational efficiency from being written in a lower-level language than FOS. Furthermore, **grpreg** calculates its coefficients in a similar manner as **glmnet** does, through running a cross-validation procedure that selects for the λ parameter that produces the lowest MSE value. Therefore, it makes sense that the coefficients produced by this λ value would yield a low MSE_{test} value.

This table provides further evidence for the accuracy of the FOS algorithm, as its group LASSO implementation outperforms the popular **grpreg** group LASSO implementation selecting for the truly influential groups.

4 Conclusion

While the FOS algorithm was outperformed by two efficient algorithms for LASSO and group LASSO, there are still some benefits that FOS can provide over what these currently used packages have to offer. In the future, it would be prudent to create an implementation of the FOS algorithm that leverages some of the computing speed that comes with code written in a lower-level language like Fortran or C. This is because while FOS and group LASSO FOS in its current form may be slower, it requires less computational power than either of the other two packages, **glmnet** and **grpreg**.

The computational efficiency of an algorithm that uses sparsity inducing penalties can be of great import in the biomedical community. One such example was alluded to in an earlier section, in the use of this algorithm for genetic expression analysis. Because FOS does not require the user to run a cross-validation procedure to find the most influential coefficients, it can be of great use in network inference.

Take the example that was mentioned in an earlier section, a dataset with the expression levels of 1000 genes as they relate to some disease. One could construct a network of correlated genes, with the nodes representing the individual genes and the edges representing a correlation between two of these genes. The construction of the correlational network would require a user to run 1000 linear regressions, explaining one gene as a function of the other 999 for each of the 1000 trials. Additionally, if it is known *a priori* that only small fraction of these genes would correlate with any other given gene, then a LASSO regression would be useful to construct this network. With the **glmnet** package, the user would have to run 1000 LASSO regressions and correspondingly 1000 cross-validation procedures to arrive at

the information necessary to construct this network. The use of an FOS algorithm, more finely-tuned than in its current state, would be able to very efficiently handle this kind of task without requiring such a heavy computing load. Furthermore, given the state of the literature on the genes included in the dataset, one could group together the genes that are of related function and add even to the computational efficiency of the procedure through the use of group LASSO FOS.

References

Breheny, Patrick, and Jian Huang. 2013. "Group Descent Algorithms for Nonconvex Penalized Linear and Logistic Regression Models with Grouped Predictors." *Statistics and Computing* 25 (2): 173–87. doi:10.1007/s11222-013-9424-2.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. 2010. "Regularization Paths for Generalized Linear Models via Coordinate Descent." *Journal of Statistical Software* 33 (1). doi:10.18637/jss.v033.i01.

Hastie, Trevor, Robert Tibshirani, and Martin Wainwright. 2015. *Statistical Learning with Sparsity: The Lasso and Generalizations*. CRC Press.

James, Gareth, Daniela Witten, Trevor J. Hastie, and Robert J. Tibshirani. 2017. *An Introduction to Statistical Learning: With Applications in R*. Springer.

Lim, Néhémy, and Johannes Lederer. 2016. "Efficient Feature Selection With Large and High-dimensional Data." *arXiv E-Prints*, September, arXiv:1609.07195.

Tibshirani, Robert. 1996. "Regression Shrinkage and Selection via the Lasso." *Journal of the Royal Statistical Society: Series B (Methodological)* 58 (1): 267–88. doi:10.1111/j.2517-6161.1996.tb02080.x.

Yuan, Ming, and Yi Lin. 2006. "Model Selection and Estimation in Regression with Grouped Variables." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68 (1): 49–67. doi:10.1111/j.1467-9868.2005.00532.x.